

Algorithm and data structures Assignment 1 experimentation

Introduction:

For both stages, using ternary search to store the input strings data character by character.

By using file input format to scan in all the data in the datafiles.

The ternary search tree takes in one string at a time and store them character by character following the order of alphabet, small goes to the left branches and big goes to the right. In the process of storing the data in the tree, one guard set to be tested is the end of string. If the current character of the word is the last character, then the weight of the whole store stored with the last character.

Using normal scanf to get the standard input as the prefix of keys, use recursive search to find the node of the last character of prefix word, if the all prefix characters exit in the tree in the same order as the prefix, the search will eventually reach the node with stores the last character of the prefix. With the node being found, use recursive travers function to traverse through the tree to all words with the same prefix.

Using linked list to store all the keywords and related weight data as in there's less chance for linked list to have memory leak problems. And it's much easier for memory allocation initiation. Besides when using selection sort, it will go through the whole list under any circumstance. In that case, it's much better with linked list.

Stage Results:

Stae1: by adding data into the tree, the tree is inserted by different alphabetic order, descending and ascending orders.

For original file:

```
kemin@DESKTOP-QR8UQGC /c/comp20005/assign1
$ autocomplete1 cities.csv output.txt < keyprefix.txt
Prefix: Melb found with 8 char comparisons
Prefix: Barcel found with 11 char comparisons
Prefix: barcel found with 9 char comparisons
Prefix: A found with 4 char comparisons
```

For descending order:

```
MINGW32:/c/comp20005/assign1
kemin@DESKTOP-QR8UQGC /c/comp20005/assign1
$ autocomplete1 testfile1.csv output.txt < keyprefix.txt
Prefix: Melb found with 14 char comparisons
Prefix: Barcel found with 21 char comparisons
Prefix: barcel found with 15 char comparisons
Prefix: A found with 8 char comparisons
```

For ascending order:

```
kemin@DESKTOP-QR8UQGC /c/comp20005/assign1
$ autocomplete1 testfile1.csv output.txt < keyprefix.txt
Prefix: Melb found with 13 char comparisons
Prefix: Barcel found with 14 char comparisons
Prefix: barcel found with 10 char comparisons
Prefix: A found with 2 char comparisons
```

Three tests on cities.csv file gives three different answers. The tree search result is varying from the input data order.

Stage 2:

With the implemented sorting algorithm, this stage has considered to have a time complexity of n^2 .

Original file with ascending order:

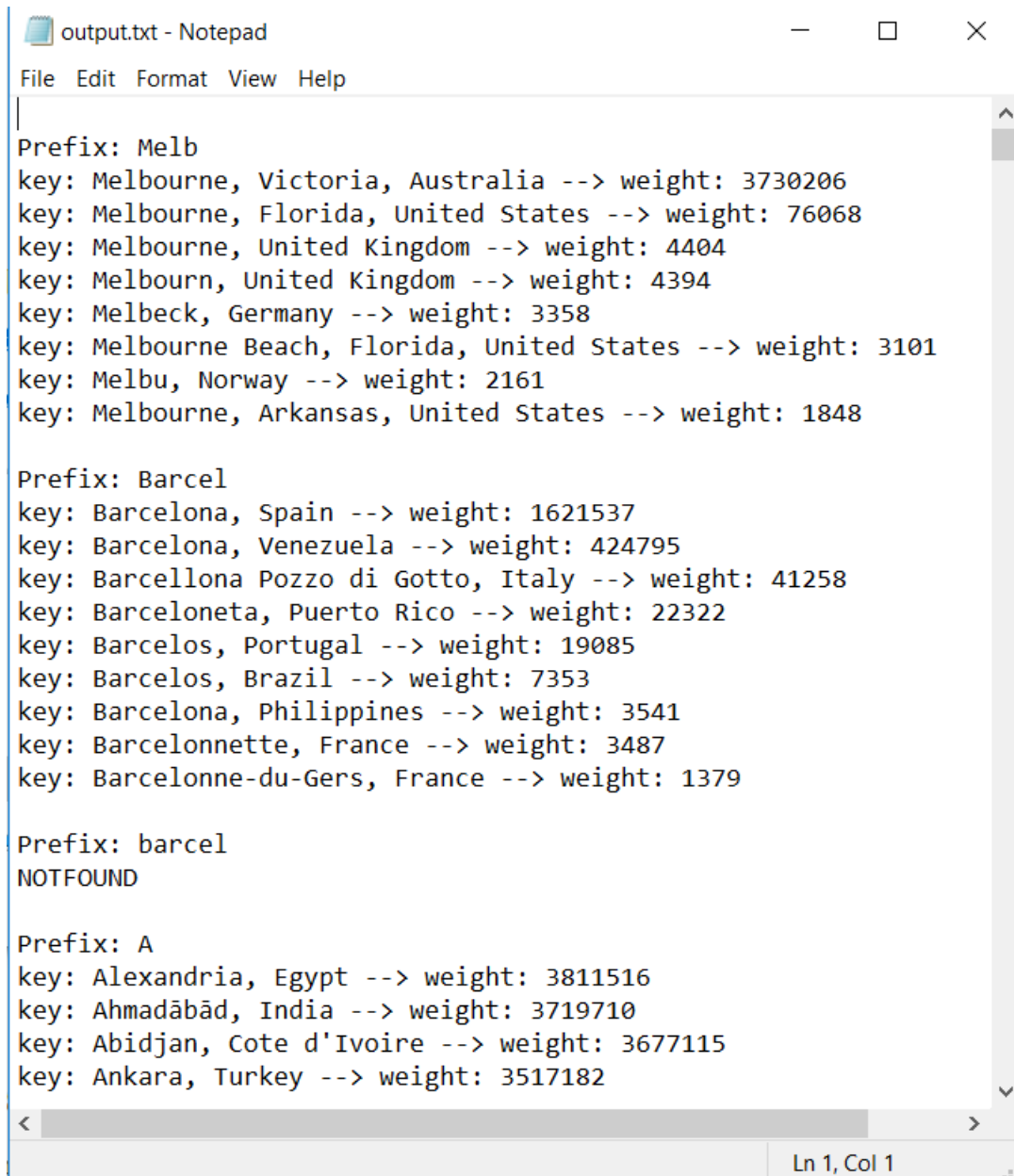
```
$ autocomplete2 cities.csv output.txt < keyprefix.txt
Prefix: Melb found with 8 char comparisons
Selection Sort: 28 weight comparison
Prefix: Barcel found with 11 char comparisons
Selection Sort: 36 weight comparison
Prefix: barcel found with 9 char comparisons
Selection Sort: 0 weight comparison
Prefix: A found with 4 char comparisons
Selection Sort: 11899881 weight comparison
```

Test file with descending order:

```
kemin@DESKTOP-QR8UQGC /c/comp20005/assign1
$ autocomplete2 testfile1.csv output.txt < keyprefix.txt
Prefix: Melb found with 18 char comparisons
Selection Sort: 28 weight comparison
Prefix: Barcel found with 20 char comparisons
Selection Sort: 36 weight comparison
Prefix: barcel found with 13 char comparisons
Selection Sort: 0 weight comparison
Prefix: A found with 1 char comparisons
Selection Sort: 11899881 weight comparison
```

Since selection sort should compare every weight in the list so the comparison number doesn't vary from the input order.

The result file is:



```
output.txt - Notepad
File Edit Format View Help

Prefix: Melb
key: Melbourne, Victoria, Australia --> weight: 3730206
key: Melbourne, Florida, United States --> weight: 76068
key: Melbourne, United Kingdom --> weight: 4404
key: Melbourn, United Kingdom --> weight: 4394
key: Melbeck, Germany --> weight: 3358
key: Melbourne Beach, Florida, United States --> weight: 3101
key: Melbu, Norway --> weight: 2161
key: Melbourne, Arkansas, United States --> weight: 1848

Prefix: Barcel
key: Barcelona, Spain --> weight: 1621537
key: Barcelona, Venezuela --> weight: 424795
key: Barcellona Pozzo di Gotto, Italy --> weight: 41258
key: Barceloneta, Puerto Rico --> weight: 22322
key: Barcelos, Portugal --> weight: 19085
key: Barcelos, Brazil --> weight: 7353
key: Barcelona, Philippines --> weight: 3541
key: Barcelonnette, France --> weight: 3487
key: Barcelonne-du-Gers, France --> weight: 1379

Prefix: barcel
NOTFOUND

Prefix: A
key: Alexandria, Egypt --> weight: 3811516
key: Ahmadābād, India --> weight: 3719710
key: Abidjan, Cote d'Ivoire --> weight: 3677115
key: Ankara, Turkey --> weight: 3517182

Ln 1, Col 1
```

Works fine with the requested prefixes.

By the theory of selection sort, giving the length of the list, the time should have a growth rate of n^2 .

Melb has 8 search results to be sorted and A has 4880 results to be sorted.

So, the number of A's result is 610 times of Melb's. The melb used to get a sorted array is 0m.00003s.

And the time A needed is around

0m.00110s ,

Which apply with the rule of selection sort.

Discussion:

Selection sort complies good with linear data structures like linked list.

The time spent by using selection sort is, on the other hand, not time efficient.